Rad sa pogledima i šablonima

Django pogled (prikaz) je posrednik između informacija (podataka) i predstavljanja tih podataka korisniku kroz šablon. Dakle, pogled je zaslužan za dostavljanje traženih podataka korisniku. Pogledi su predstavljeni kao Python funkcije ili klase.

Pogledi kao klase

Prvobitni dizajn Djanga je podrazumevao samo korišćenje funkcija unutar pogleda, a objektno orijentisani princip je dodat tek kasnije. Sa tim dodavanjem se mogućnost ponovnog korišćenja istog koda povećala, a sa njom i produktivnost.

Pogledi kao funkcije

Ove funkcije prihvataju web zahtev kao argument i vraćaju web odgovor kao povratnu vrednost te funkcije. Odgovor može biti sve od HTML koda, preko XML koda, PDF dokumenata, slika, pa do grešaka i preusmeravanja. Na ovaj način možemo brzo postaviti svoju aplikaciju. Problem nastaje tek u velikim Django projektima, gde na ovaj način podižemo kompleksnost kodu. U našem primeru, koji se nadovezuje na projekat iz prethodne nastavne jedinice ćemo, radi lakšeg objašnjavanja i razumevanja, koristiti poglede sa funkcijama.

Implementacija pogleda

Implementaciju svog prvog pogleda ćemo zasnovati na već kreiranom projektu first_project iz prethodne nastavne jedinice. Koristeći komandnu liniju, pozicioniraćemo se u folder first_project, u kojem se ujedno nalazi i fajl manage.py, i pokrenuti komandu za kreiranje nove aplikacije pod nazivom book_library:

```
python manage.py startapp book_library
```

Nakon izvršenja ove komande, u projektnom direktorijumu se pojavljuje još jedan folder, pod imenom book_library, koji odgovara našoj aplikaciji. U njemu ćemo naći views.py fajl, u koji ćemo dodati funkciju index(), koja će biti zadužena za ispis teksta *Book library* kada korisnik otvori početnu stranu našeg sajta. Pre nastavka je potrebno otvoriti settings.py fajl i u listi INSTALLED_APPS dodati novi element čija će vrednost biti tipa string i to – ime naše aplikacije, kako bi Django znao da smo dodali još jednu aplikaciju:

Izgled INSTALLED_APPS liste:

```
INSTALLED_APPS = [
      'django.contrib.admin',
      'django.contrib.auth',
      'django.contrib.contenttypes',
      'django.contrib.sessions',
      'django.contrib.messages',
      'django.contrib.staticfiles',
      'book_library',
]
```

Odmah po kreiranju naše aplikacije, bez ikakvih izmena, views.py izgleda ovako:

```
from django.shortcuts import render
# Create your views here.
```

Dakle, jedino što postoji u tom fajlu po kreiranju je naredba za uvoz render funkcije, zadužene za slanje šablona korisniku, o čemu će biti reči kasnije. Pošto smo rekli da pogledi bazirani na funkcijama imaju za ulazni parametar request – zahtev korisnika ka serveru, a za povratnu vrednost response – odgovor servera klijentu, naš views.py fajl nakon implementacije index() funkcije izgleda ovako:

book_library/views.py nakon naše implementacije:

```
from django.shortcuts import render
from django.http import HttpResponse
books = [{'title':'The Picture of Dorian Gray','year':'1890'},
      {'title':'Pride and Prejudice','year':'1813'},
      {'title':'The Adventures of Tom Sawyer ', 'year':'1875'},
      {'title':'The Raw Youth', 'year':'1875'},
      {'title':'Twelve Years a Slave ', 'year':'1853'},
      {'title':'Hamlet', 'year':'1603'}]
# Create your views here.
def index(request):
      return HttpResponse("<h1>Book library</h1>")
```

Takođe, pored implementacije ove funkcije, dodali smo i listu rečnika sa imenima knjiga i godinom izlaska koja će nam kasnije biti potrebna. Prvo što vidimo u odnosu na onaj bazični views.py kreiran po nastanku aplikacije je da je uvezena klasa HttpResponse. Ona nam služi da odgovor, odnosno povratnu vrednost index() funkcije, pretvorimo u HttpResponse objekat. Pomoću ovog objekta podatke koje želimo poslati pretvaramo u odgovor. Pomoću HttpResponse() objekta možemo postaviti i željena polja u zaglavlju. Opširnije uputstvo za korišćenje ovog objekta možete pronaći u zvaničnoj dokumentaciji.

Konstruktoru ove klase smo prosledili jednostavan deo HTML koda – h1 tag sa tekstom *Book library*, koji će se i ispisati korisniku na ekranu.

Nakon što smo dodali željenu funkcionalnost fajlu views.py, potrebno je podesiti naš Django projekat da na početnoj strani i ispiše *Book library*. Ovo možemo uraditi na dva načina:

Podešavanje URL-a iz projektnog direktorijuma:

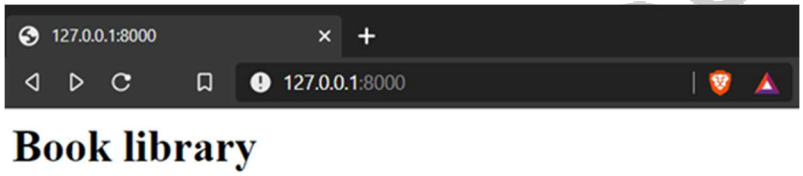
Za ovaj način potrebno je otvoriti urls.py fajl, koji se nalazi u projektnom direktorijumu (first_project/urls.py), koji nakon povezivanja url-a kojem će pogled-funkcija index() odgovarati izgleda ovako:

first_project/urls.py kod:

```
from django.contrib import admin
from django.urls import path, include
from book_library import views
urlpatterns = [
      # path('', admin.site.urls),
      path('', views.index, name = 'index')
]
```

U odnosu na urls.py fajl koji je generisan prilikom prvobitnog kreiranja projekta, uviđamo par razlika. Prva je da smo pored path funkcije uvezli i include funkciju, koja će nam kasnije poslužiti. Prvobitnu liniju path('', admin.site.urls) smo zakomentarisali, jer nam sada nije potrebna. Takođe smo, sada iz naše aplikacije book_library, uvezli i views.py fajl, kako bismo pristupili index() funkciji. Lista urlpatterns, koja je takođe kreirana prilikom kreiranja samog projekta, popunjava se elementima koji predstavljaju funkcijski poziv path(). Ovoj funkciji se prosleđuju minimalno dva parametra, gde je prvi tipa string i sadrži putanju po kojoj će, kada joj korisnik pristupi, funkcija iz pogleda obraditi zahtev, a drugi obavezni argument funkcije path() je upravo funkcija iz pogleda koju smo definisali. U ovom slučaju je to views.index. O detaljnijem mapiranju putanja će biti reči u sledećoj nastavnoj jedinici. Pošto u našem primeru želimo da na početnoj stranici ispišemo tekst *Book library*, kao okidač views.index() funkcije smo postavili putanju '', koja predstavlja početnu stranu.

Pošto smo snimili oba fajla (i first_project/urls.py i book_library/views.py), možemo pokrenuti server komandom python manage.py runserver 127.0.0.1:8000. Ako u pretraživaču otvorimo ovu adresu, dobićemo sledeći ekran:



Slika 13.1. Rezultat implementacije naše index() funkcije

Podešavanje URL-a iz direktorijuma aplikacije

U prethodnom primeru smo videli kako mapirati URL iz projektnog urls.py fajla, ali je to moguće i kombinacijom projektnog i aplikacijskog urls.py fajla. Ovo je posebno dobro odraditi na većim Django projektima, jer olakšava organizaciju i mapiranje velikog broja linkova i deli ih po aplikacijama. Naime, radi uštede vremena, iskopiraćemo urls.py iz projektnog foldera u folder naše aplikacije. Promene koje su potrebne da bi i na ovaj način korisnik koji otvori naš sajtu dobio ispis *Book library* na ekranu su sledeće:

first_project/urls.py:

```
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
      # path('', admin.site.urls),
      # path('', views.index, name = 'index')
      path('', include('book_library.urls'))
]
```

book_library/urls.py

```
from django.contrib import admin
from django.urls import path, include
# from book_library import views
from . import views
urlpatterns = [
      # path('', admin.site.urls),
      path('', views.index, name = 'index')
]
```

Prva razlika koju vidimo u izmenjenom first_project/urls.py fajlu je da sada umesto eksplicitnog pozivanja index() funkcije iz našeg pogleda u path() pozivu koristimo include() funkciju. Zato i ne uvozimo više views.py fajl iz aplikacijskog direktorijuma. Njen argument je tipa string i predstavlja putanju do urls.py fajla koji zapravo želimo da koristimo u ovom slučaju. Sa druge strane, u book_library/urls.py uvozimo views.py fajl, koji se nalazi u istom direktorijumu kao i u book_library/urls.py i njega implementiramo slično kao u prvom slučaju. Nakon pokretanja našeg servera, dobićemo isti prikaz na ekranu. Da bi se jasnije videla prednost ovog načina, poslužićemo se prostim primerom. Recimo da u first_project/urls.py fajlu ispišemo ovakav poziv path() funkcije:

```
path('about/', include('book_library.urls'))
```

a da u book_library/urls.py imamo ovakvu organizaciju urlpatterns liste:

```
urlpatterns = [
      path('', views.index, name = 'index'), # line 1
      path('contact/', views.index_2, name = 'index_2'), # line 2
      path('blog/', views.index_3, name = 'index_3'), # line 3
      ]
```

Ovo znači da bi se korisniku, kada bi otvorio about/ stranu (mapiranje izvršeno iz first_project/urls.py fajla), prikazala funkcija index(); ako bi korisnik pristupio stranici about/contact/, prikazala bi mu se funkcija index_2(), dok bi mu se u trećem slučaju, ako bi pristupio stranici about/blog/, prikazala funkcija index_3. Zato se može reći da su, pri ovakvom pristupu, linkovi u first_project/urls.py bazni linkovi koji se kasnije, u zavisnosti od aplikacije, nadograđuju.

Django šabloni (templates)

Do sada smo koristli poglede za prikaz podataka. Ali, kako sajt baziran na Django radnom okviru raste, takav način postaje težak, i za implementaciju i za održavanje se koriste šabloni kako bi se odvojila logika od same forme tih podataka. U šablonima je predefinisan izgled tih podataka i njihov redosled, a samo je potrebno proslediti im podatke. Prvo što ćemo izmeniti jeste naša index() funkcija, i to tako da umesto HttpResponse objekta vraća šablon:

```
def index(request):
      # return HttpResponse("<h1>Book library</h1>")
      return render(request, 'index.html', {'books':books})
```

Kao što možemo videti iz zakomentarisane linije, razlika između prethodnog pristupa i ovog je u korišćenju render() funkcije, čija je namena generisanje odgovora iz šablona i slanje klijentu, koja za parametre uzima:

- request – klijentski zahtev upućen serveru koji je neophodan zbog daljeg konteksta prilikom kreiranja šablona namenjenom upravo tom klijentu;
- index.html – šablonski fajl;
- {'books':books} – rečnik sa vrednostima koje će render() funkcija iskoristiti kako bi popunila polja u .index.html fajlu.

Pre nego što kreiramo šablonski index.html fajl, u istom direktorijumu naše aplikacije ćemo kreirati i folder templates, u koji ćemo smestiti upravo ovaj fajl sa sledećim HTML kodom:

Index.html fajl:

```
<!DOCTYPE html>
<html>
	<head>
		<title>Book Library</title>
	</head>
	<body>
		<h1>Our book repository</h1>
		<div>
			<ul>
		    {% for book in books %}
				<li>
					<h2>Title: {{ book.title }} </h2>
					<p> Year: {{ book.year}} </p>
				</li>
		    {% endfor %}
			</ul>
		</div>
	</body>
</html>
```

Ono što se razlikuje od samog HTML koda su redovi koji počinju jednostrukim ili dvostrukim vitičastim zagradama, koji pomažu u prebacivanju podataka iz pogleda u šablon, a o kojima će kasnije u lekciji biti reči. Sačuvati fajl kao index.html.

Takođe, važno je napomenuti da se za napredniju konfiguraciju šablona koristi settings.py fajl, i to lista TEMPLATES. U našem primeru, kako bismo Djangu pokazali u kom direktorijumu da traži naš index.html fajl, potrebno je izmeniti upravo tu listu, polje DIRS, gde nakon izmene cela lista izgleda ovako:

TEMPLATES lista:

```
TEMPLATES = [
	{
	'BACKEND': 'django.template.backends.django.DjangoTemplates',
	'DIRS': [os.path.join(BASE_DIR,'templates')],
	'APP_DIRS': True,
	'OPTIONS': {
```

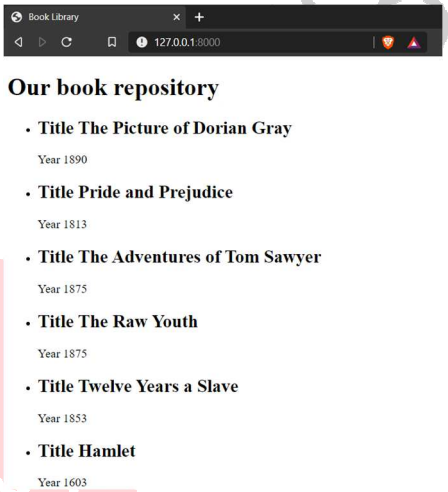
```
                'context_processors': [
                    'django.template.context_processors.debug',
                    'django.template.context_processors.request',
                    'django.contrib.auth.context_processors.auth',
                    'django.contrib.messages.context_processors.messages',
                ],
            },
            },
    ]
```

Dodavanjem linije ’ os.path.join(BASE_DIR,'templates')’ smo ’rekli’ Đangu da za svaku od aplikacija u listi ’INSTALLED_APPS’, kada je reč o šablonima – da ih potraži u folderu ’templates’.

Napomena:

Ako u bilo kom trenutku prilikom pokretanja dobijemo *ImportError: no module named os* grešku, treba otvoriti settings.py fajl i na njegovom početku dodati liniju import os.

Nakon izmene konfiguracije u settings.py fajlu, kao i izmene views.py fajla, možemo pokrenuti primer i na početnoj strani će nas sačekati izgled nešto drugačiji od prethodnog:



Slika 13.2. Početna stranica sa korišćenjem šablona

Django šablonski jezik

Kako bismo nesmetano koristili Python kod i u HTML-u, Django nudi sopstveni šablonski jezik, sa kojim smo se već susreli koristeći index.html fajl (jednostruke i dvostruke vitičaste zagrade). Djangov šablonski jezik nam omogućava da čitavu HTML stranu generišemo na serveru i kao takvu, bez ikakvog drugog Python koda, pošaljemo klijentu, čime dobijamo na sigurnosti i efikasnosti generisanja šablona. Što se Django šablonskog jezika tiče, postoji četiri tipa sintakse:

Pitanje

U kojoj podlisti liste TEMPLATES dodajemo informaciju o lokaciji šablonskih fajlova?

- **'DIRS'**
- 'Context_processors'
- Ova informacija ne dodaje se u listu TEMPLATES

Objašnjenje:

Tačan odgovor je da informaciju o dodavanju šablonskih fajlova upisujemo u DIRS podlistu liste TEMPLATES.

Promenljive

Sintaksa: {{ variable_name }} – okružene dvostrukim vitičastim zagradama.

Promenljive nam služe radi ispisivanja neke vrednosti, pa će se tako u našem index.html primeru umesto <h2>Title {{ book.title }} </h2> ispisati: „Title The Picture of Dorian Gray”. S tim, pošto smo u našem primeru prosledili listu rečnika, ključu rečnika, umesto Python sintakse key[`value’], pristupili notacijom tačke.

Tagovi/kontrolna logika

Sintaksa: {% tag_name %} – okružene jednostrukim vitičastim zagradama i znakom za procenat.

Pomoću ovih tagova implementiramo logiku u šablonu. U našem primeru smo u index.html fajlu koristli for petlju koja počinje linijom {% for book in books %} i završava linijom {% endfor %}. Na taj način smo omogućili iteraciju kroz promenljivu books koju smo prosledili pomoću index() funkcije.

Lista svih dostupnih tagova se može naći u zvaničnoj dokumentaciji.

Filteri

Sintaksa: {{ variable_name | filter_name }} – okružena dvostrukim vitičastim zagradama, a pored imena promenljive prosleđuje se znak |, nakon kojeg sledi ime filtera. Moguće je i povezivanje filtera, tako da je i ovakav primer moguć: {{ variable_name|escape|linebreaks }}.

Korišćenjem ovih filtera možemo menjati i transformisati vrednost promenljive direktno u šablonu.

Lista svih dostupnih filtera se može naći u zvaničnoj dokumentaciji.

Komentari

Sintaksa: {% comment %} – okružena jednostrukim vitičastim zagradama i znakom %. Ovaj deo koda se mora zatvoriti {% endcomment %} linijom, jer će sve između ove dve linije prevodilac Django šablonskog jezika preskočiti. Kao i u Pythonu, komentari nam omogućavaju dodatno pojašnjavanje linija i sekcija koda. Ovi komentari će biti prisutni samo u našim fajlovima; u izvornom HTML kodu koji stigne do klijenta se neće videti.

Upotreba šablona je moguća i u mapiranju URL putanja na našem sajtu, uz korišćenje projektnog urls.py fajla (first_project/urls.py), s tim što se u tom slučaju ništa od sintakse šablonskog jezika neće izvršiti, pa bi nakon otvaranja početne stranice našem korisniku na ekranu bilo ispisano samo *Our book respository* (iako i dalje imamo mapirane URL-ove u book_library/urls.py fajlu), a urlpatterns lista bi izgledala ovako:

first_project/urls.py:

```
from django.contrib import admin
from django.urls import path, include
from django.views.generic import TemplateView
urlpatterns = [
      # path('', admin.site.urls),
      # path('', views.index, name = 'index')
      # path('', include('book_library.urls')),
      path('',TemplateView.as_view(template_name='index.html'))
]
```

Glavna razlika između ovog načina prikazivanja podataka korisniku i prethodnog načina – preko funkcije u pogledu – jeste ta da uvozimo TemplateView klasu pomoću koje mapiramo čitav index.html fajl na scenario kada korisnik pristupa početnoj strani. Ovaj način nam pomaže u situacijama kada imamo već pripremljen statični .html fajl u kojem nema šablonske sintakse (već čist HTML/CSS/JS).

Više informacija o šablonskom jeziku može se naći u zvaničnoj dokumentaciji.

Rezime

- Django pogled (prikaz) je posrednik između informacija (podataka) i predstavljanja tih podataka korisniku kroz šablon.
- Pogledi su predstavljeni kao Python funkcije ili klase.
- U fajlu settings.py, lista INSTALLED_APPS nam služi za dodavanje naših aplikacija trenutnom projektu.
- Klasa HttpResponse nam služi da odgovor, odnosno povratnu vrednost index() funkcije pretvorimo u HttpResponse objekat. Pomoću ovog objekta, podatke koje želimo da pošaljemo pretvaramo u odgovor. Pomoću HttpResponse() objekta možemo postaviti i željena polja u zaglavlju;
- Argument funkcije include() je tipa string i predstavlja putanju do urls.py fajla koji zapravo želimo da koristimo prilikom mapiranja željene putanje.
- Namena render() funkcije je generisanje odgovora iz šablona i slanje klijentu; ona za parametre uzima trenutni klijentski zahtev, ime šablona, kao i rečnik čiji su ključevi imena promenljivih koje koristimo u šablonu, a vrednosti promenljive iz Pythona.

- Djangov šablonski jezik nam omogućava da čitavu HTML stranu generišemo na serveru i kao takvu, bez ikakvog drugog Python koda, pošaljemo klijentu, čime dobijamo na sigurnosti i efikasnosti generisanja šablona.